

Implementing Dense Linear Algebra Algorithms on Multi-Core Processors Using Dataflow Execution Model

Jakub Kurzak
Jack Dongarra
University of Tennessee

SIAM Parallel Processing for Scientific Computing
MS18: Dataflow 2.0: The Re-emergence and Evolution
of Dataflow Programming Techniques for Parallel Computing
March 13, 2008

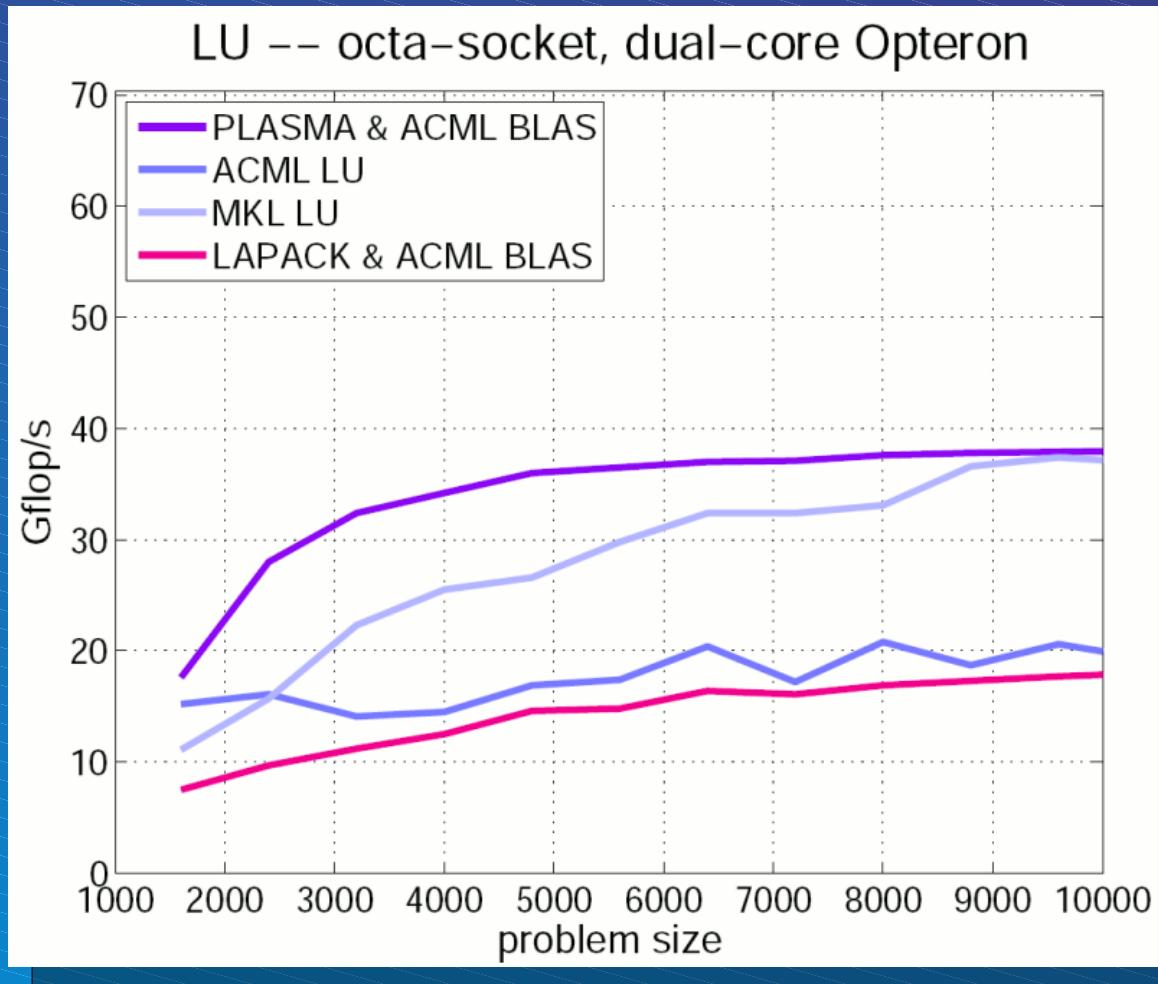
New Class of Algorithms

- ◆ LU, QR, Cholesky, one-sided, factorizations
 - ◆ Inspired by out-of-memory algorithms
 - ◆ Processing of input by small tiles
 - ◆ Block data layout
 - ◆ Fine granularity
 - ◆ **Suitable for dynamic scheduling**
-
- ◆ LAPACK Working Note 190
 - ◆ LAPACK Working Note 191
 - ◆ LAPACK Working Note 184

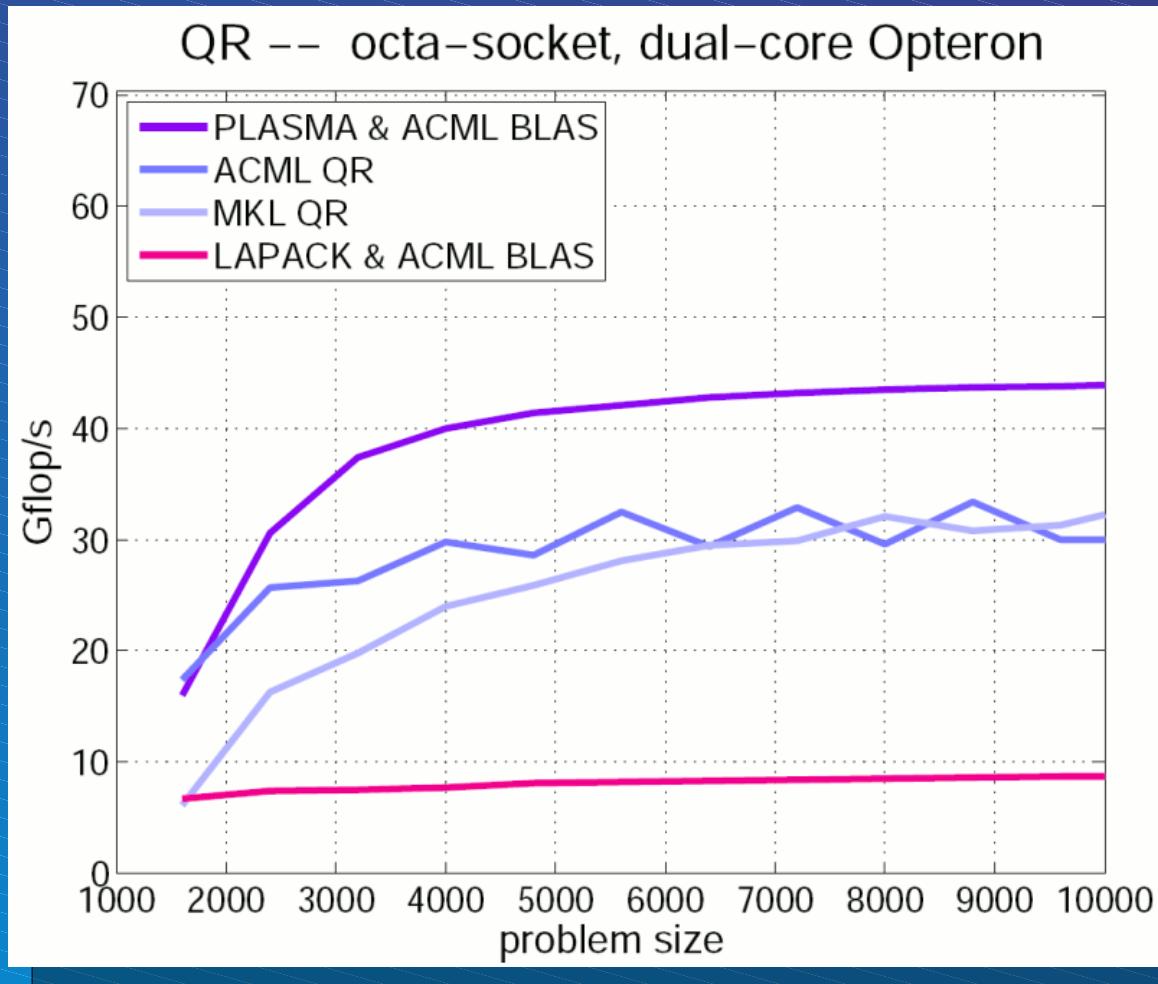
New Class of Algorithms

- ◆ Classic multi-core (x86)
 - ◆ Faster than MKL
 - ◆ Faster than ACML
 - ◆ Way faster than LAPACK
- ◆ Exotic multi-core (CELL)
 - ◆ Extremely fast
 - ◆ No alternatives

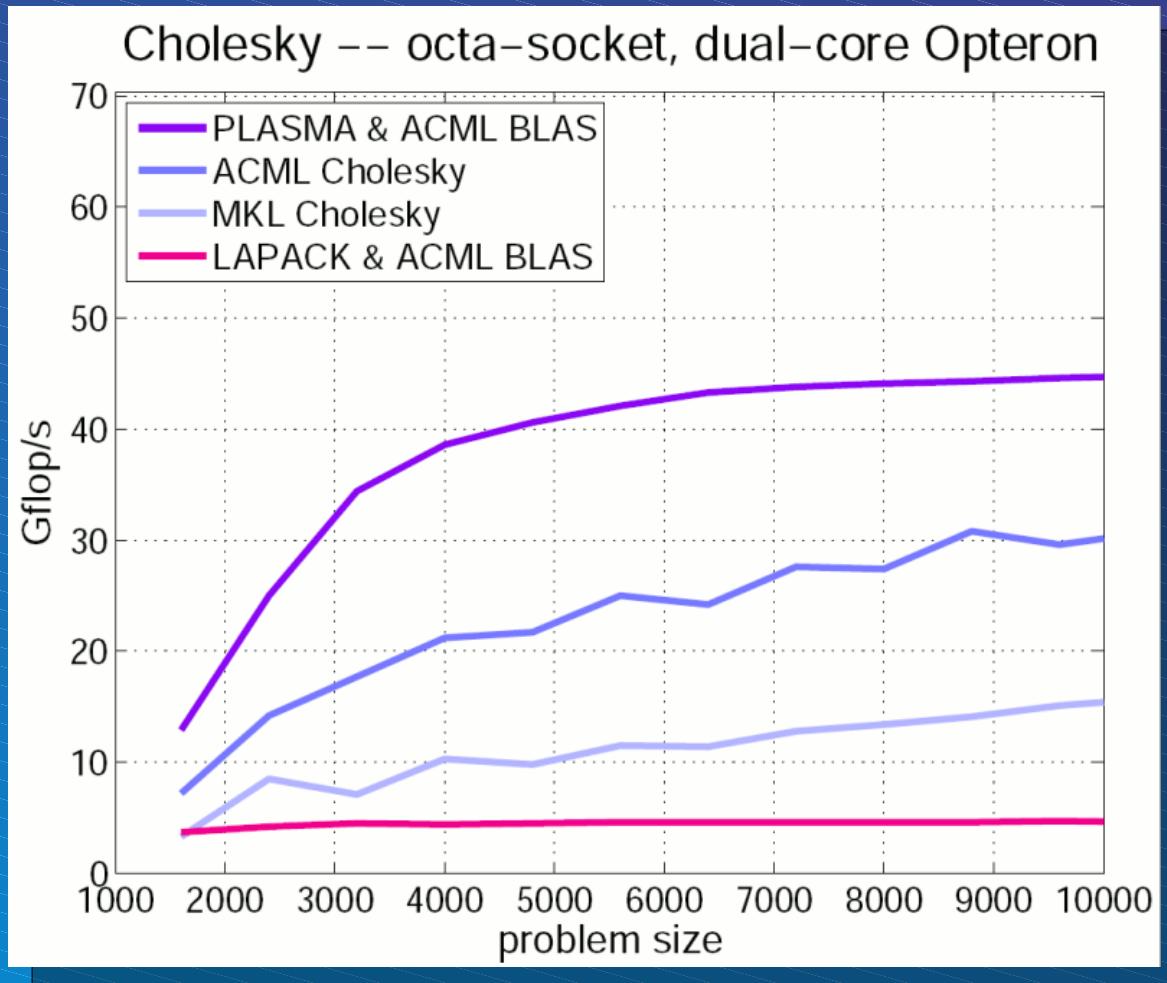
New Algorithms – LU on x86



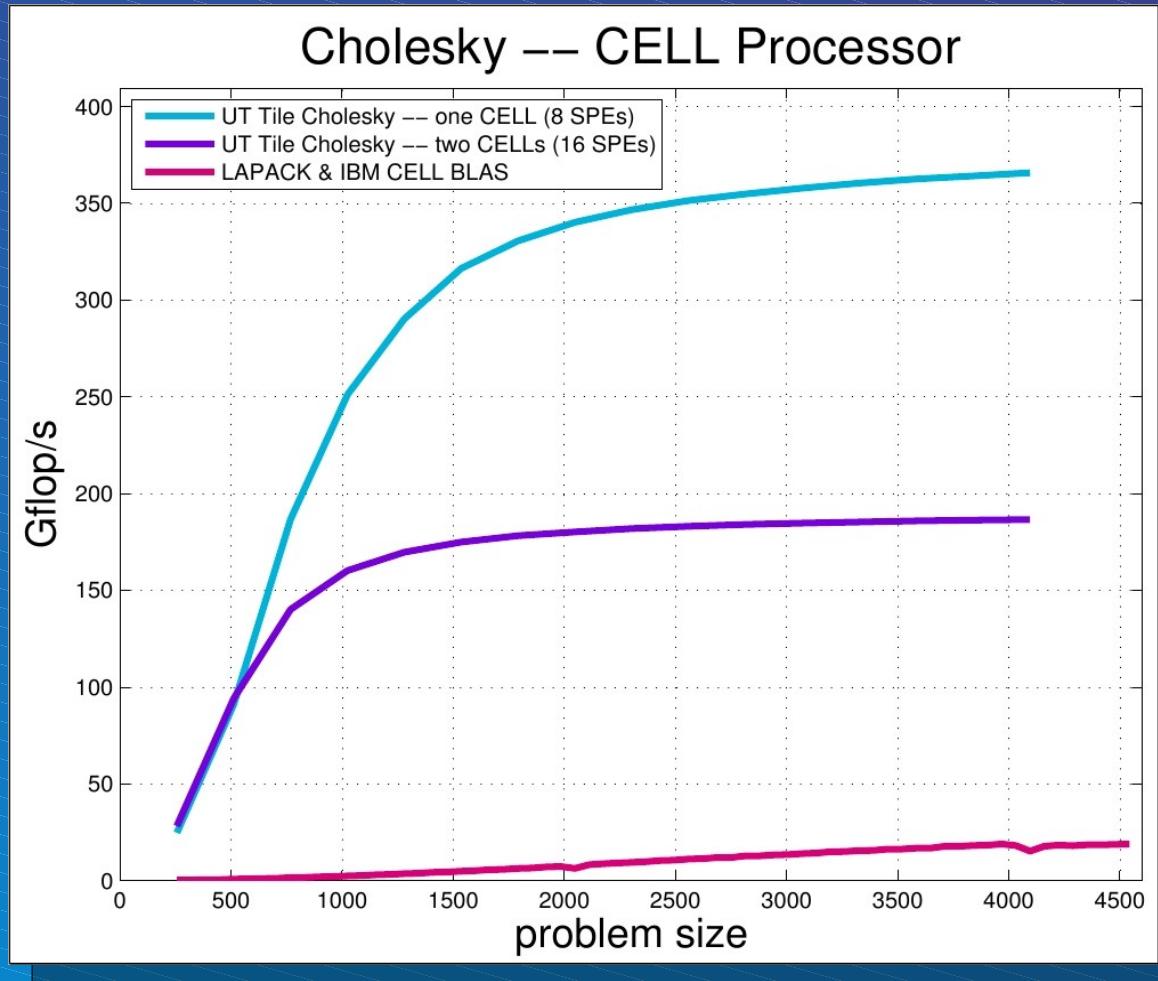
New Algorithms – QR on x86



New Algorithms – Cholesky on x86



New Algorithms – Cholesky on the CELL

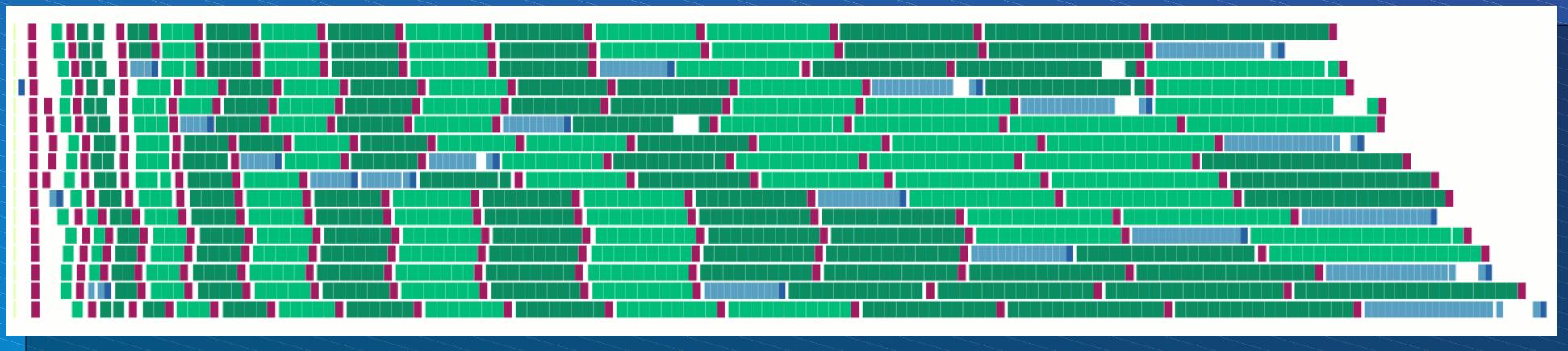


Going “Out of Sync”

- ◆ CELL Cholesky – 8 cores



- ◆ CELL Cholesky – 16 cores



PLASMA Framework

- ◆ Don't reinvent the wheel
 - ◆ SCHEDULE
 - ◆ HeNCE
 - ◆ Parametrized DAGs (Cosnard, Jeannot)
 - ◆ Dataflow Systems (Simulink, Gedae)
 - ◆ Systolic Arrays

PLASMA Goals (WHAT)

- ◆ **Scalability**
 - ◆ Massive parallelism
- ◆ **Performance**
 - ◆ Efficient runtime
- ◆ **Portability**
 - ◆ Cluster / MPP
 - ◆ SMP / CMP
 - ◆ Out-of-memory / CELL

PLASMA Principles (HOW)

- ◆ Data-driven (dynamic) execution
- ◆ Explicit expression of parallelism
- ◆ Implicit communication
- ◆ Component architecture
 - ◆ “Language”
 - ◆ Runtime
 - ◆ Tools
 - ◆ Dataflow visualization
 - ◆ DAG visualization
 - ◆ Profiling
 - ◆ Tracing
 - ◆

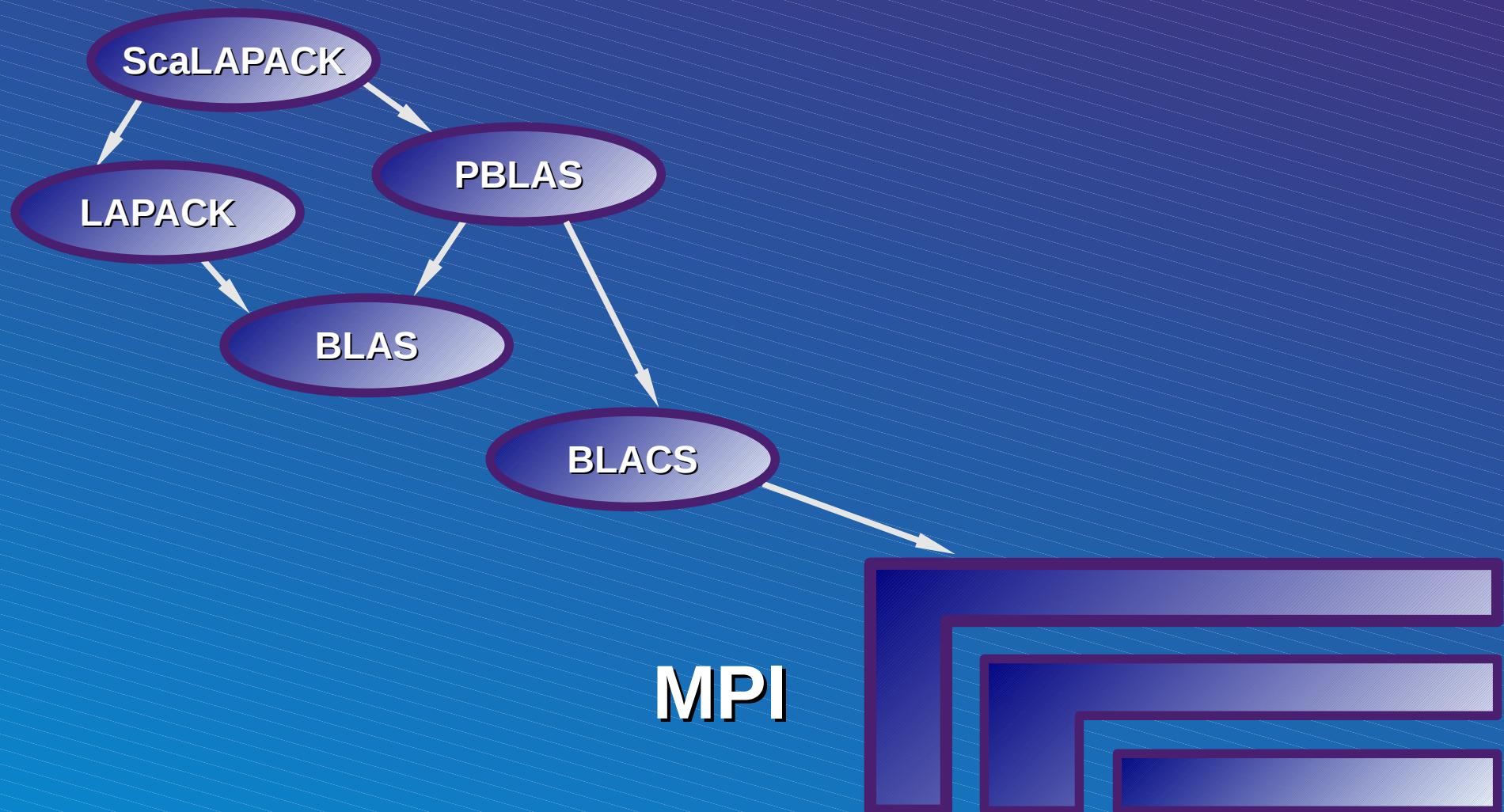
There is no such thing as autoparallelization.

BTW,

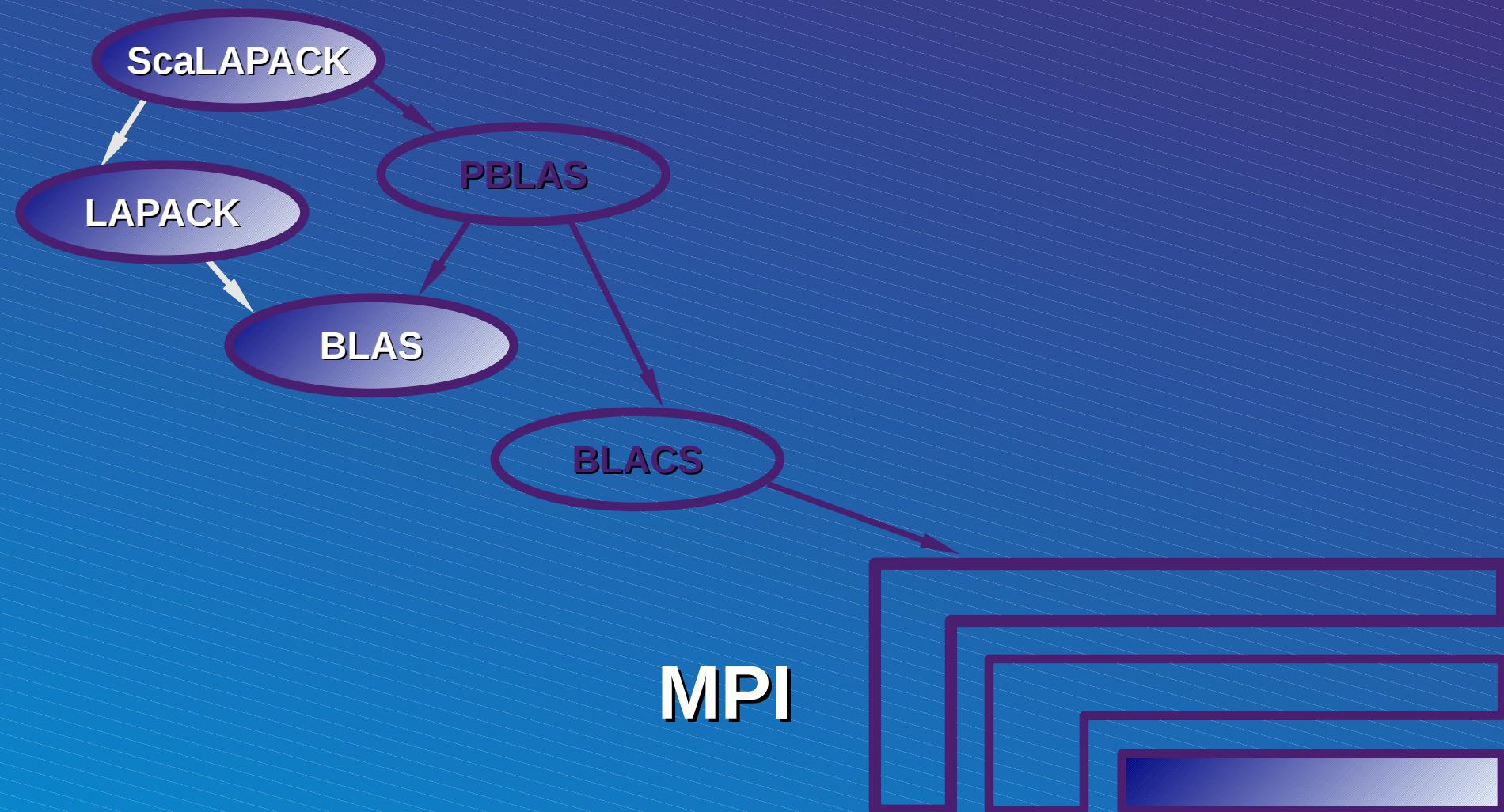
There is no such thing as auto SIMD'zation.

(see LAPACK Working Note 189)

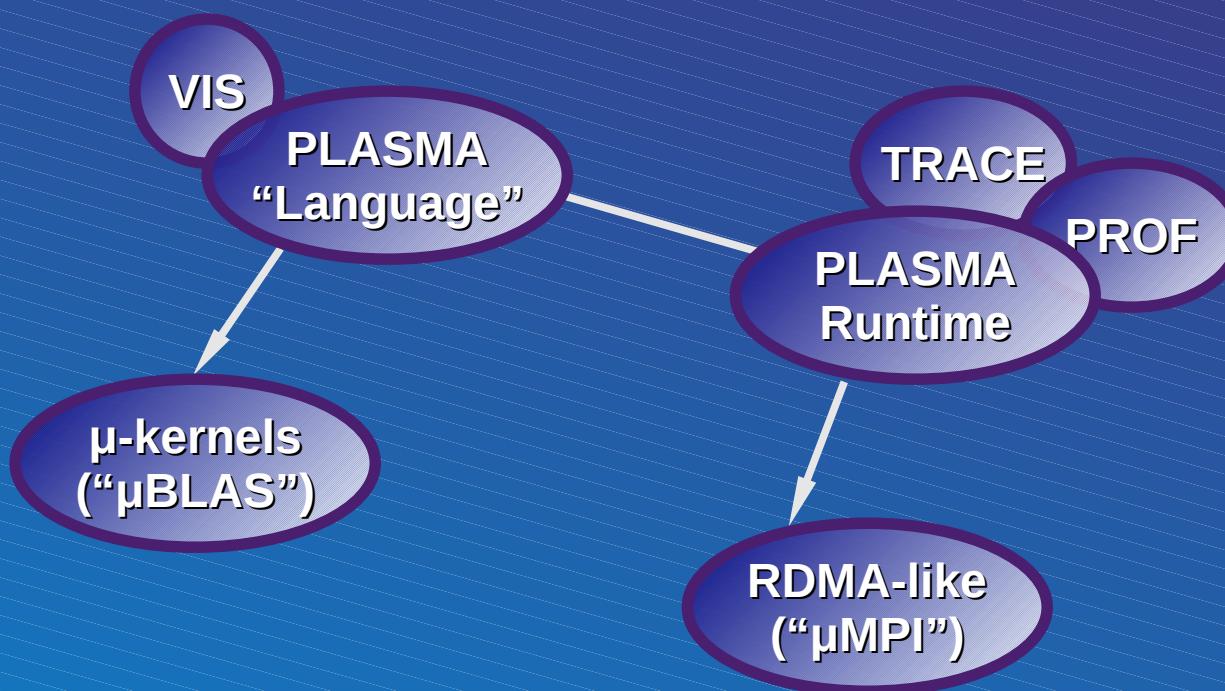
Current Infrastructure



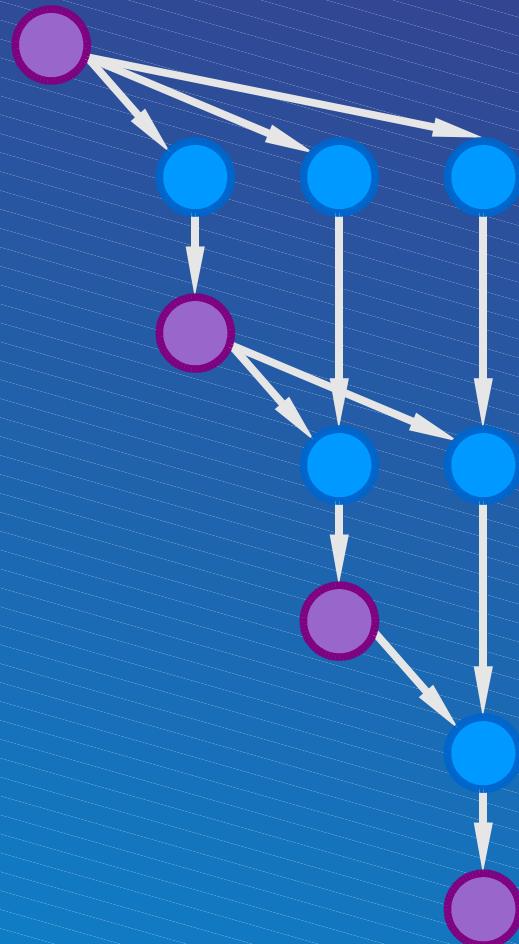
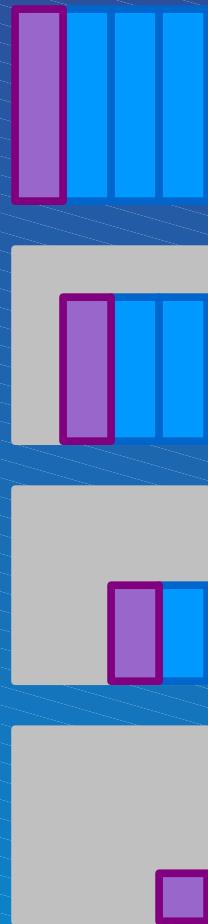
Current Infrastructure



Target Infrastructure



DAG / Task Graph



- ◆ view computation as a DAG
- ◆ use dependency / data driven scheduling
- ◆ pursue the critical path

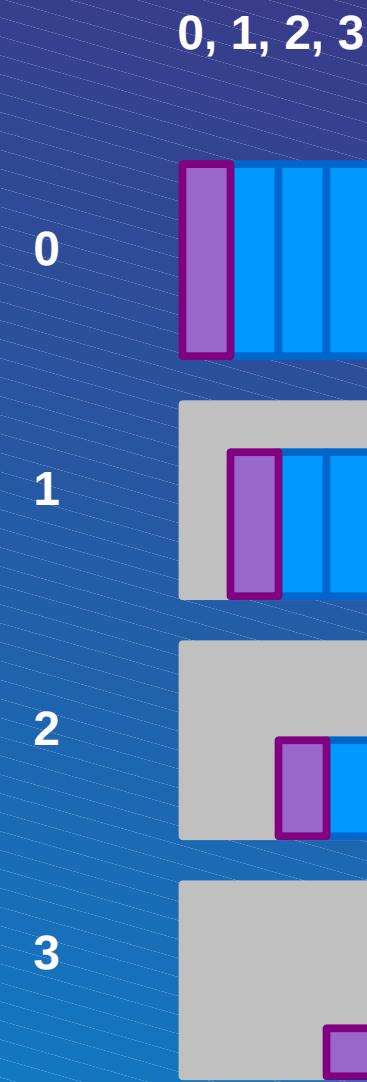
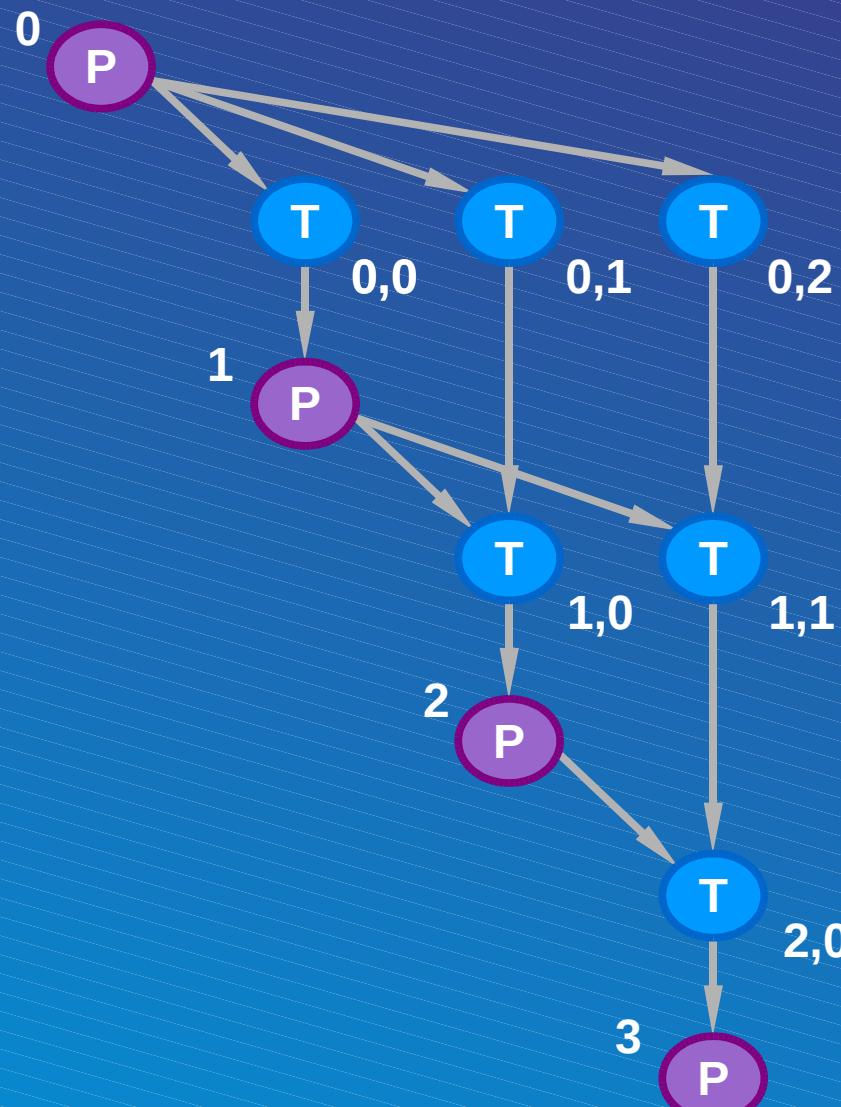
DAG / Task Graph

- ◆ DAGs can be huge
 - ◆ Difficult / impossible to store
 - ◆ Difficult / impossible to manage
- ◆ I don't need to build the DAG
- ◆ I need the ability to traverse the DAG

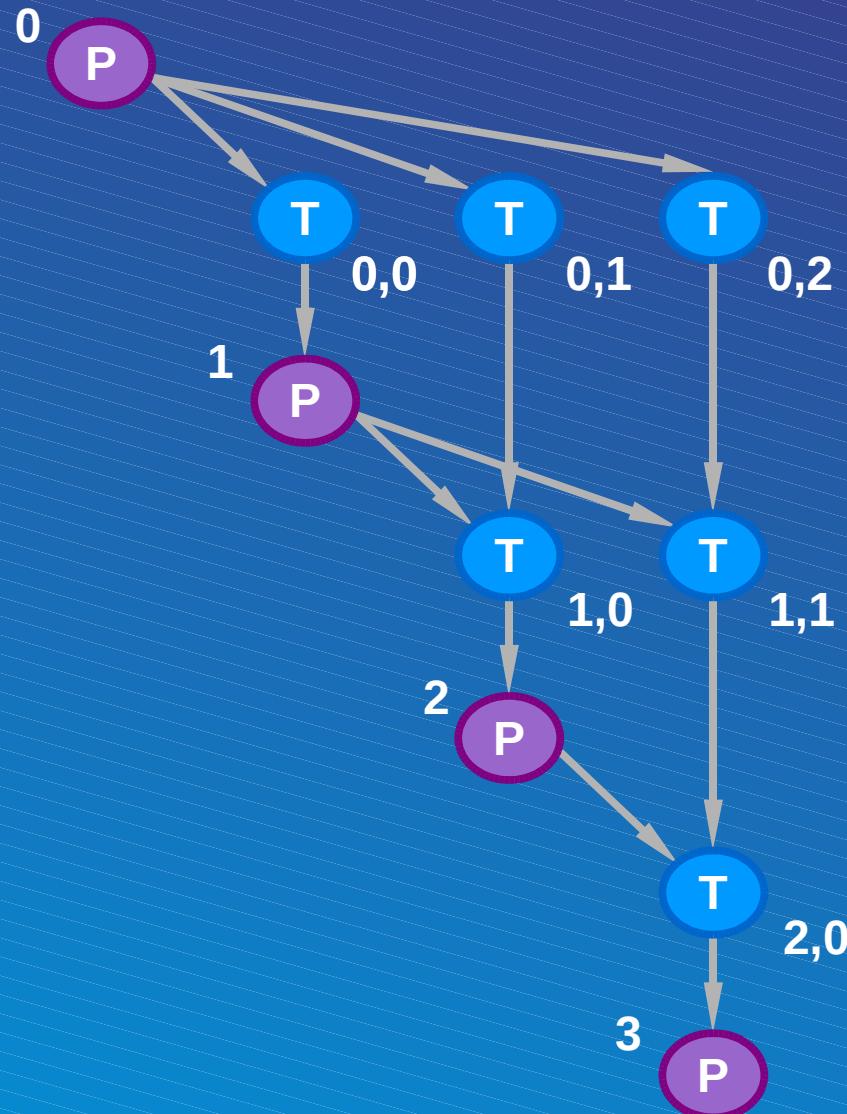
DAG / Task Graph

- ◆ Producer Task
 - ◆ knows where its output goes
- ◆ Consumer Task
 - ◆ knows where its input comes from
- ◆ Nobody knows the DAG
- ◆ Parents know their children
- ◆ Children know their parents

Task ID / Task Coordinates



Task Definition



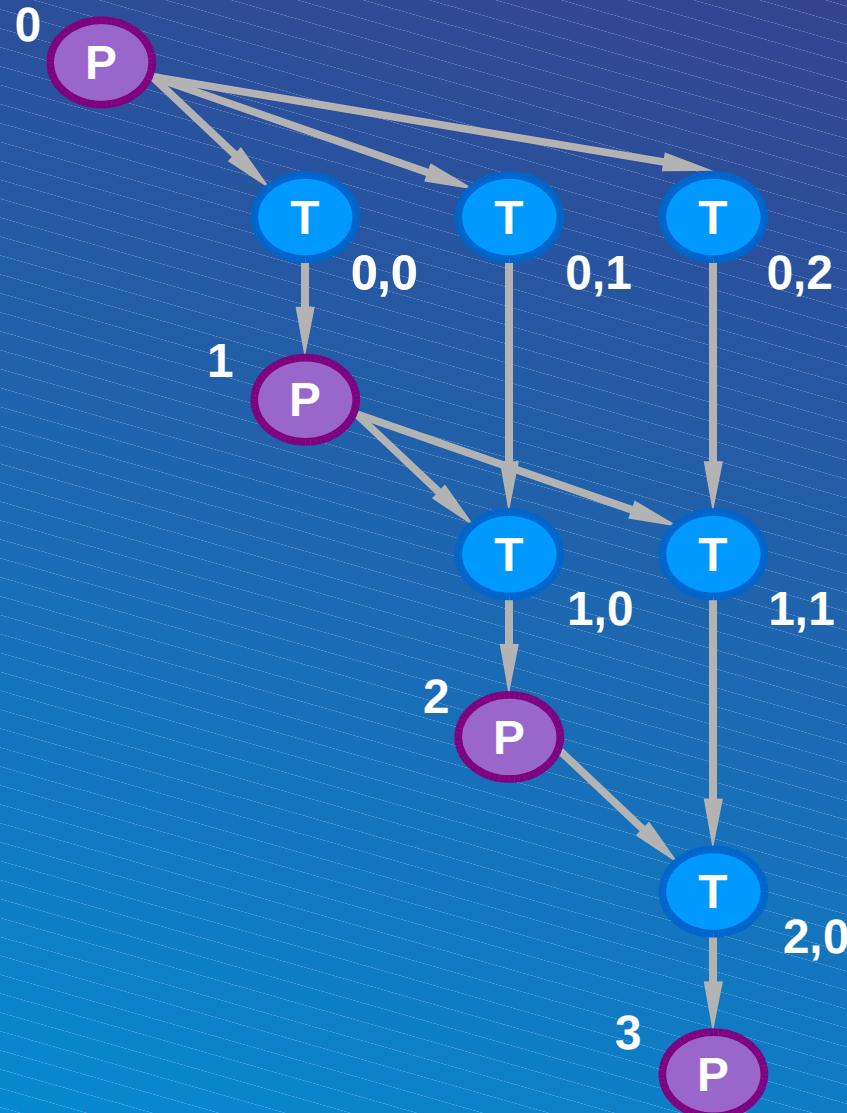
// Task Name
P

// Execution Space
 $i = 0 \dots N$

// Parallel Partitioning
 $i \% Ncores = coreID$

// Parameters
input $\leftarrow T(i - 1, 0)$
output $\rightarrow T(i, 0 \dots i)$

Task Definition



// Task Name

T

// Execution Space

$i = 0 \dots N - 1$

$j = 0 \dots N - 1 - i$

// Parallel Partitioning

$i \% \text{Ncores} = \text{coreID}$

// Parameters

$\text{input } 1 \leftarrow P(i)$

$\text{input } 2 \leftarrow T(i - 1, j + 1)$

$\text{output } \rightarrow T(i + 1, j - 1)$

```

// Name
POTRF(k)

// Execution space
k = 0..SIZE

// Parallel partitioning
k % GRIDrows == rowRANK
k % GRIDcols == colRANK

// Parameters
INOUT T <- (k == 0) ? IN(k, k) : T SYRK(k, k)
    -> T TRSM(k, k+1..SIZE)
    -> OUT(k, k)

// Body
lapack_spotrf(lapack_lower, NB, T, NB, &info);

```

```

// Name
GEMM(k, m, n)

// Execuiton space
k = 0..SIZE
m = k+1..SIZE
n = 0..k-1

// Parallel partitioning
m % GRIDrows == rowRANK
k % GRIDcols == colRANK

// Parameters
IN A <- C TRSM(n, k)
IN B <- C TRSM(n, m)
INOUT C <- (n == 0) ? IN(m, n) : C GEMM(k, m, n-1)
    -> (n == k-1) ? C TRSM(k, m) : C GEMM(k, m, n+1)

// Body
cblas_sgemm(CblasColMajor,
            CblasNoTrans, CblasTrans,
            NB, NB, NB,
            1.0, B, NB, A, NB, 1.0, C, NB);

```

```

// Globals
GRIDrows, GRIDcols, NB

```

```

// Name
SYRK(k, n)

// Execution space
k = 0..SIZE
n = 0..k-1

// Parallel partitioning
k % GRIDrows == rowRANK
k % GRIDcols == colRANK

// Parameters
IN A <- C TRSM(n, k)
INOUT T <- (n == 0) ? IN(k, k) : T SYRK(k, n-1)
    -> (n == k-1) ? T POTRF(k) : T SYRK(k, n+1)

// Body
cblas_ssyrk(CblasColMajor,
            CblasLower, CblasNoTrans,
            NB, NB, 1.0, A, NB, 1.0, T, NB);

```

```

// Name
TRSM(k, m)

// Execution space
k = 0..SIZE
m = k+1..SIZE

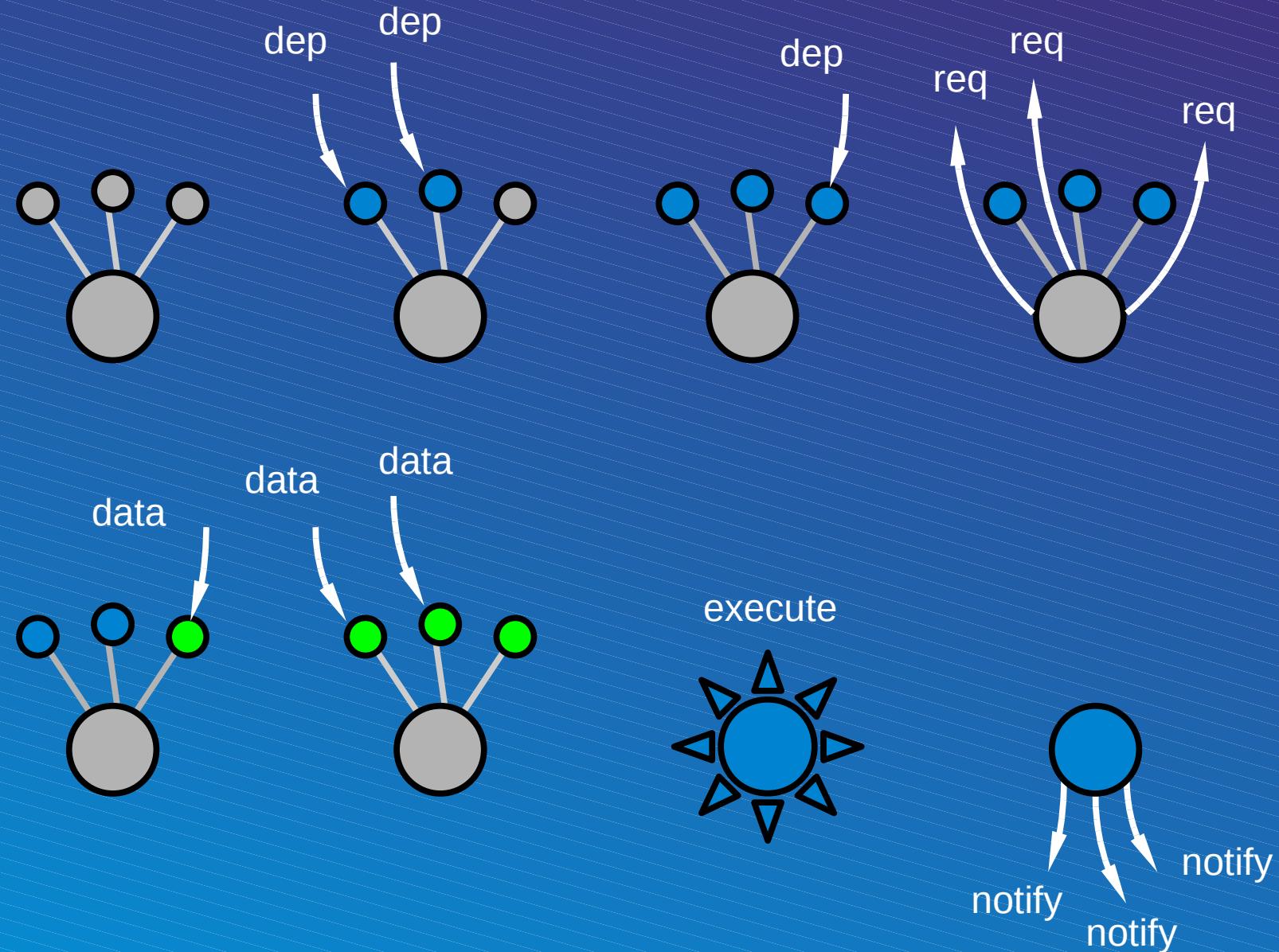
// Parallel partitioning
m % GRIDrows == rowRANK
k % GRIDcols == colRANK

// Parameters
IN T <- T POTRF(k)
INOUT C <- (k == 0) ? IN(m, k) : C GEMM(k, m, k-1)
    -> A GEMM(m, m+1..SIZE, k)
    -> B GEMM(k+1..m-1, m, k)
    -> A SYRK(m, k)
    -> OUT(k, m)

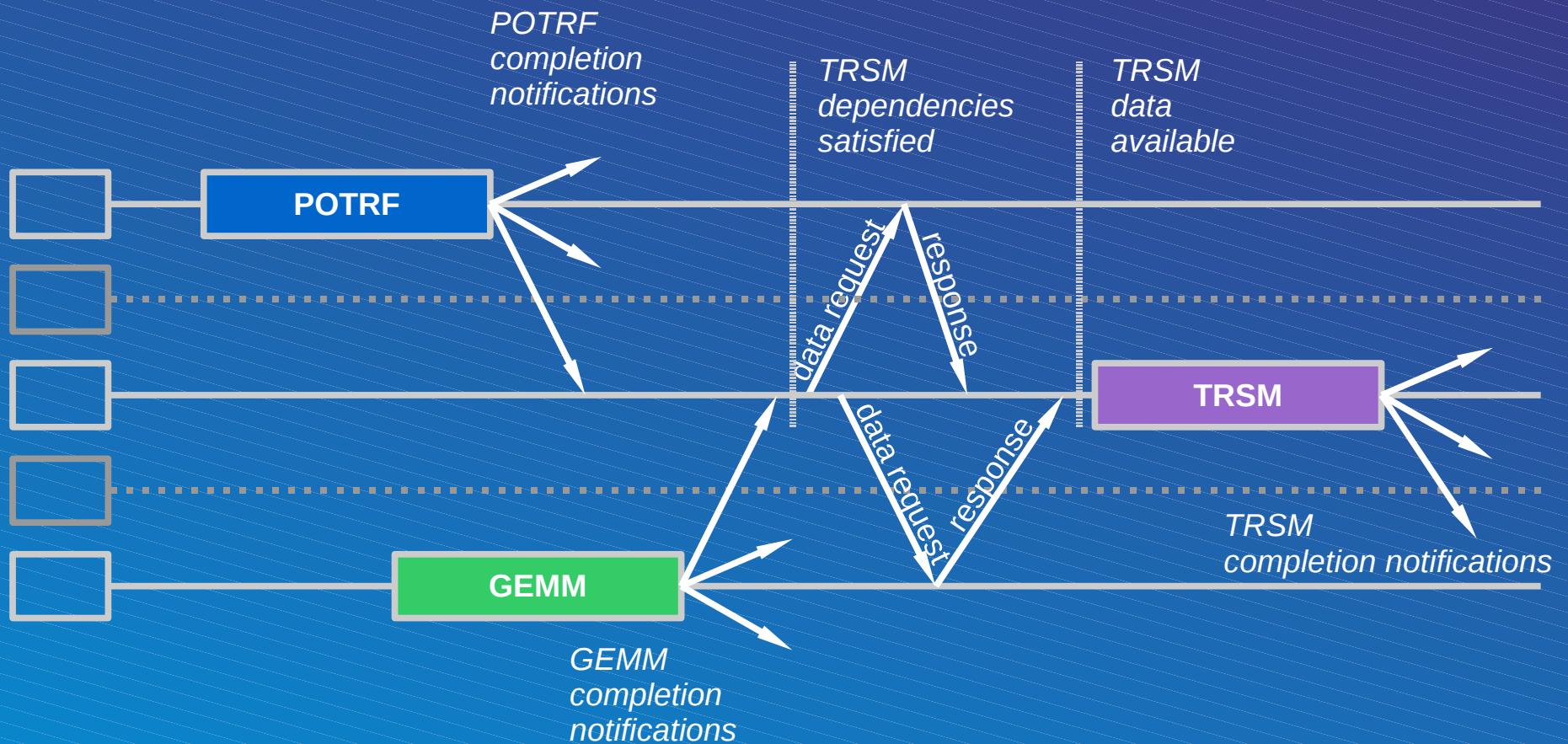
// Body
cblas_strsm(CblasColMajor,
            CblasRight, CblasLower,
            CblasTrans, CblasNonUnit,
            NB, NB, 1.0, T, NB, B, NB);

```

Task Life-Cycle

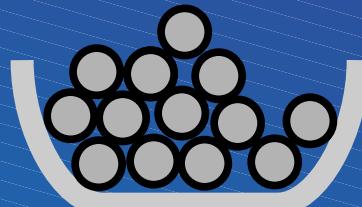


Task Life-Cycle

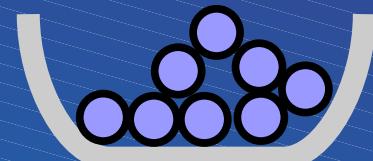


Task Life-Cycle

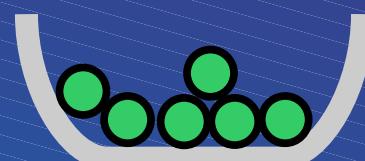
- ◆ no dependencies satisfied



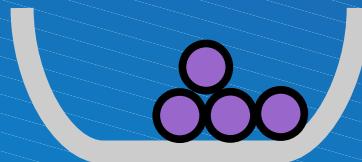
- ◆ all dependencies satisfied
- ◆ some data delivered
- ◆ waiting for all data



- ◆ some dependencies satisfied
- ◆ waiting for all dependencies



- ◆ all data delivered
- ◆ waiting for execution



Compilation

- ◆ Existing compiler technology
 - ◆ task body – standard C / Fortran
 - ◆ formulas – C / Fortran expressions
 - ◆ metadata – human readable ASCII

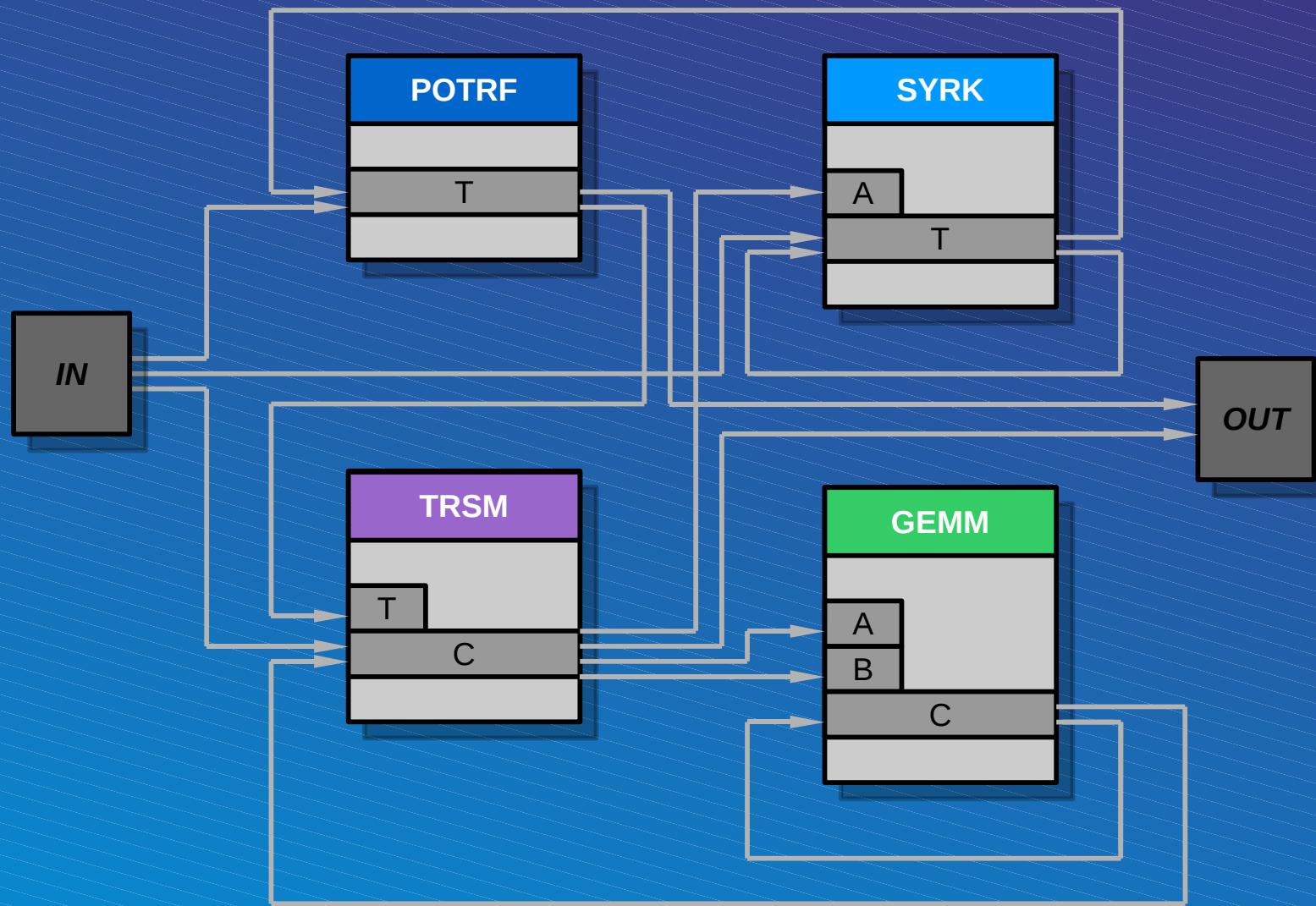
Runtime

- ◆ Scheduler
 - ◆ management of task bins
- ◆ Communication system
 - ◆ schared mem – pointer aliasing
 - ◆ distrib mem – messaging
 - ◆ out-of-memory – mix

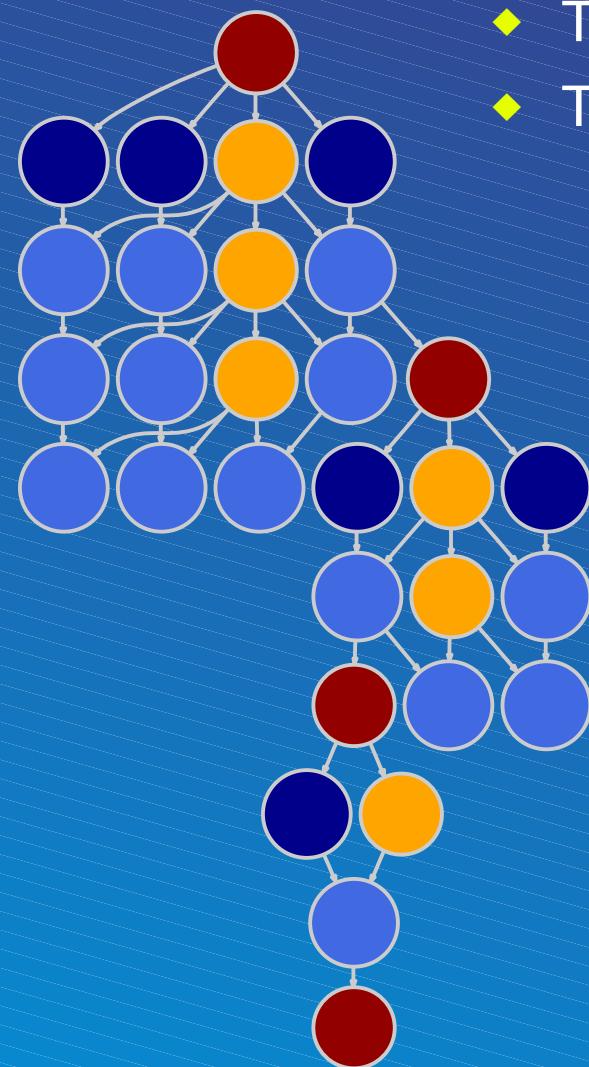
Visual Tools

- ◆ Standalone ASCII code
- ◆ Little / No error checking
- ◆ Visual tools to assist development
 - ◆ plot dataflow diagram
 - ◆ instantiate small DAG

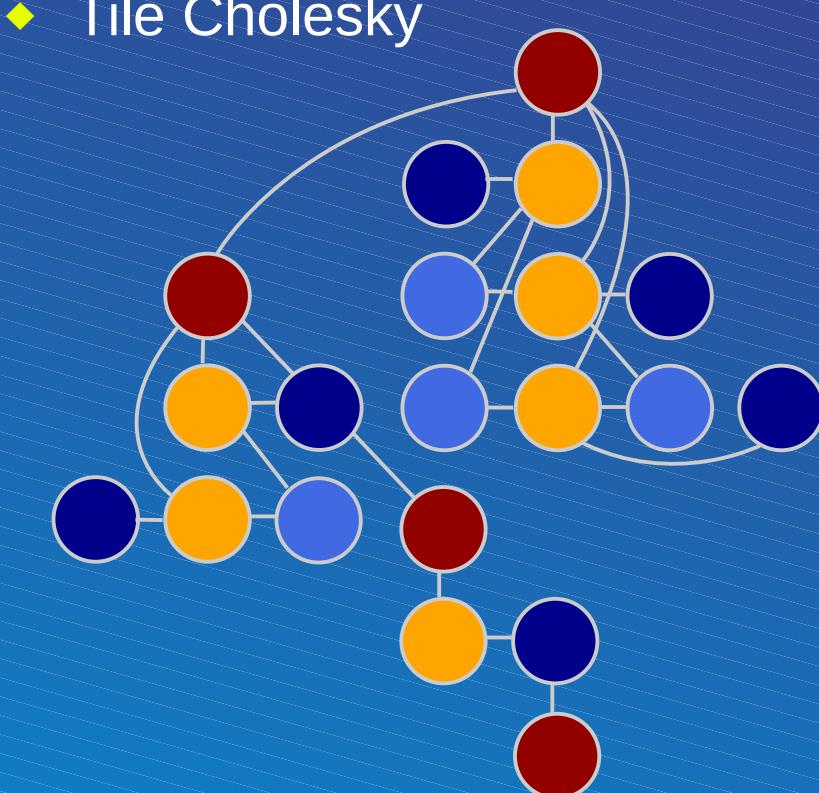
Dataflow Visualization



Task Graph Visualization

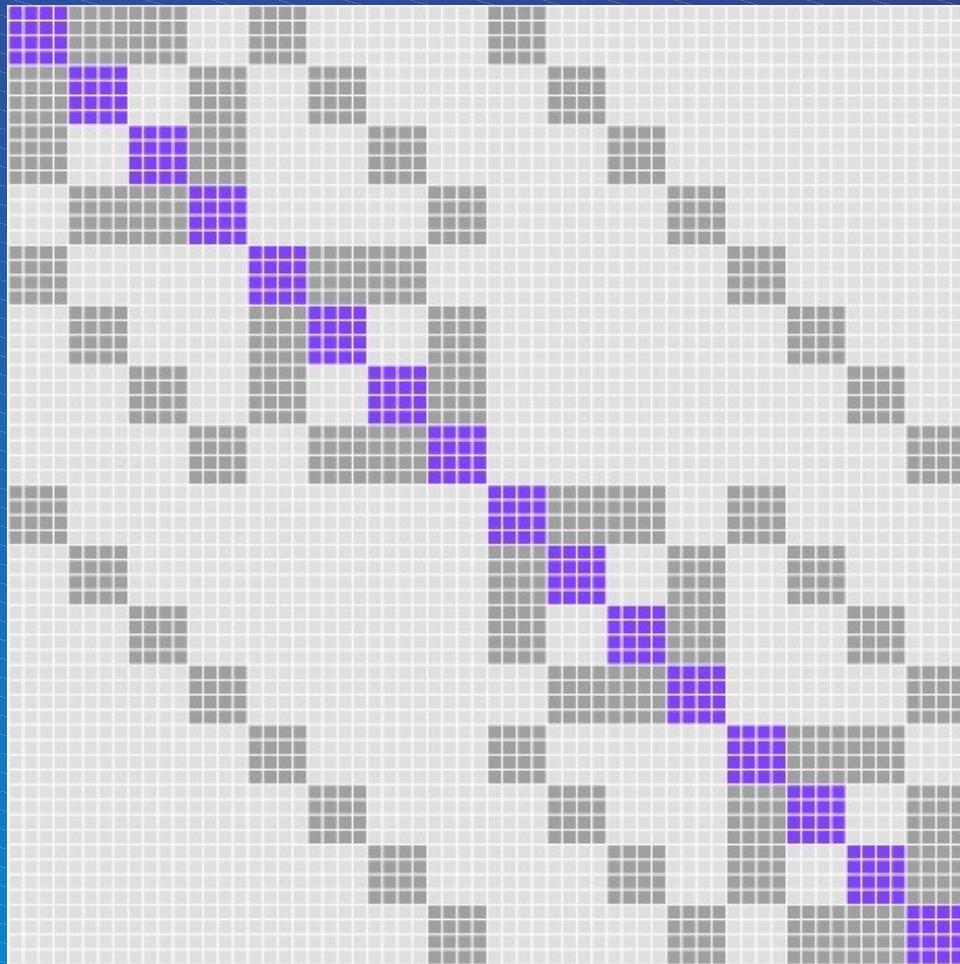


- ◆ Tile LU
 - ◆ Tile QR



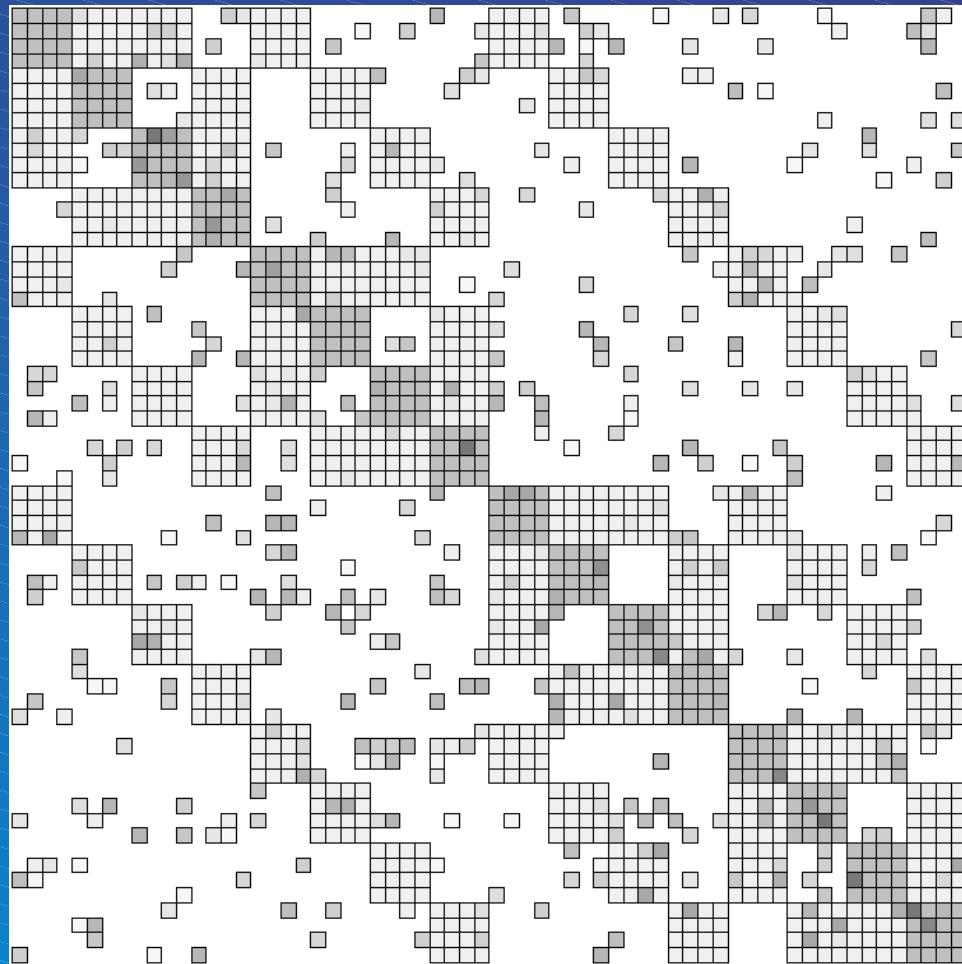
- ## ◆ Tile Cholesky

Connectivity / Topology



- ◆ 4D cube
- ◆ quad-core

Connectivity / Topology



communication matrix should resemble connectivity matrix

Thank You !

Questions ?